Algorithms of Scientific Computing

Space-Filling Curves and their Applications in Scientific Computing

Michael Bader

Summer Term 2012

Sequentialising Multi-dimensional Data

Examples of multi-dimensional data structures:

- Matrices
- Image data (images, tomographic data, movies, ...)
- discretisation meshes (to discretise mathematical models in physics/...; PDE, etc.)
- Coordinates (often used in connection with graphs)
- tables (also in data bases)
- in computational finance and financial mathematics: “baskets” of stocks/options/...

Sequentialising Multi-dimensional Data (2)

Typical algorithms and operations:

- traversal (update/processing of all data; simulation meshes, e.g.)
- matrix operations (linear algebra, etc.)
- sequentialisation (e.g. to store data on discs or in main memory)
- partitioning of data (for parallelisation or in divide-and-conquer algorithms)
- sorting of data (to simplify further operations)
- in general: nested loops

```
for i from 1 to n do
    for j from 1 to m do          ...
```

Demands on Efficient Sequentialisation

Effective Sequentialisation:

- unique numbering $\Rightarrow$ requires bijective mapping
- sequentialisation without “holes” (for data structures, e.g.)

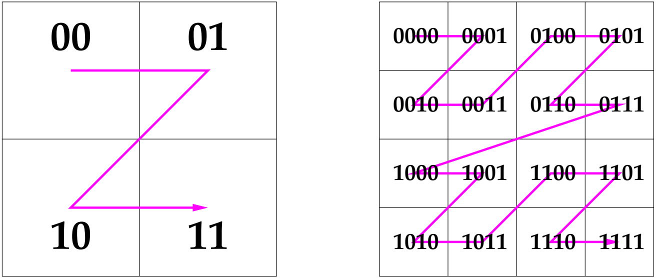
Efficient Sequentialisation:

- preserve neighbourship properties $\Rightarrow$ data locality
- fast, simple index computation
- “smoothness”, stability vs. small changes
- dimensional symmetry (no fast or slow dimensions)
- “clustering” of data

Application Examples

- **range queries** in image and raster data bases
- **image browsing** and **image search** in image collections
- heuristical approaches for graph-based algorithms (nearest neighbour, traveling salesman)
- collision detection
- **parallelisation** of data
- efficient use of **cache memory** (in simulations, e.g.)

Start: Morton Order / Cantor's Mapping



Questions:

- Can this mapping lead to a **contiguous** "curve"?
- i.e.: Can we find a **continuous** mapping?
- and: Can this continuous mapping fill the entire square?

What is a Curve?

Definition (Curve)

As a **curve**, we define the image $f_*(\mathcal{I})$ of a *continuous* mapping $f\colon \mathcal{I} \to \mathbb{R}^n$.
$x = f(t), t \in \mathcal{I}$, is called **parameter representation** of the curve.

With:

- $\mathcal{I} \subset \mathbb{R}$ and $\mathcal{I}$ is compact, usually $\mathcal{I} = [0, 1]$.
- the **image** $f_*(\mathcal{I})$ of the mapping f_* is defined as $f_*(\mathcal{I}) := \{f(t) \in \mathbb{R}^n \mid t \in \mathcal{I}\}$.
- $\mathbb{R}^n$ may be replaced by any Euklidian vector space (norm & scalar product required).

What is a Space-filling Curve?

Definition (Space-filling Curve)

Given a mapping $f\colon \mathcal{I} \to \mathbb{R}^n$, then the corresponding curve $f_*(\mathcal{I})$ is called a **space-filling curve**, if the Jordan content (area, volume, ...) of $f_*(\mathcal{I})$ is larger than 0.

Comments:

- assume $f\colon \mathcal{I} \to \mathcal{Q} \subset \mathbb{R}^n$ to be **surjective** (i.e., every element in $\mathcal{Q}$ occurs as a value of f);

  then, $f_*(\mathcal{I})$ is a space-filling curve, if the area (volume) of $\mathcal{Q}$ is positive.
- if the domain $\mathcal{Q}$ has a smooth boundary, then there can be **no bijective mapping** $f\colon \mathcal{I} \to \mathcal{Q} \subset \mathbb{R}^n$, such that $f_*(\mathcal{I})$ is a space-filling curve
  (theorem: E. Netto, 1879).

History of Space-Filling Curves

1877: Georg Cantor finds a bijective mapping from the unit interval $[0, 1]$ into the unit square $[0, 1]^2$.

1879: Eugen Netto proves that a **bijective** mapping $f\colon \mathcal{I} \to \mathcal{Q} \subset \mathbb{R}^n$ can not be continuous (i.e., a curve) at the same time (as long as $\mathcal{Q}$ has a smooth boundary).

1886: rigorous definition of **curves** introduced by Camille Jordan

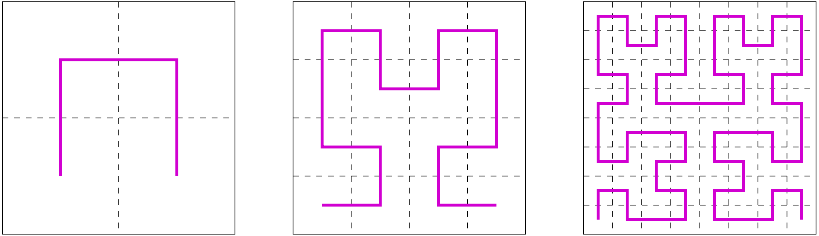
1890: Giuseppe Peano constructs the first space-filling curves.

1890: Hilbert gives a geometric construction of Peano's curve; and introduces a new example – the Hilbert curve

1904: Lebesgue curve

1912: Sierpinski curve

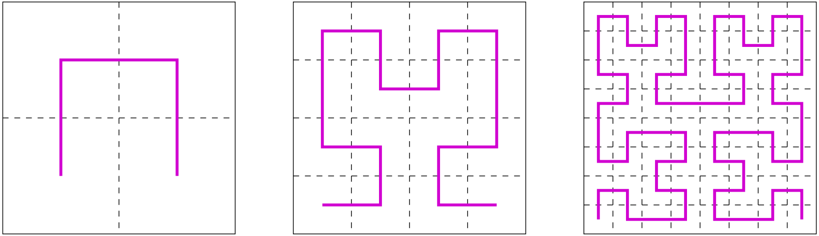
Construction of the Hilbert curve



Iterations of the Hilbert curve:

- start with an iterative numbering of 4 subsquares
- combine four numbering patterns to obtain a twice-as-large pattern
- proceed with further iterations

Construction of the Hilbert curve



Recursive construction of the **iterations**:

- split the quadratic domain into 4 congruent subsquares
- find a space-filling curve for each subdomain
- join the four subcurves in a suitable way

Definition of the Hilbert Curve's Mapping

Definition: (Hilbert curve)

- each parameter $t \in \mathcal{I} := [0, 1]$ is contained in a sequence of intervals

$$\mathcal{I} \supset [a_1, b_1] \supset \ldots \supset [a_n, b_n] \supset \ldots,$$

  where each interval result from a division-by-four of the previous interval.
- each such sequence of intervals can be uniquely mapped to a corresponding sequence of 2D intervals (subsquares)
- the 2D sequence of intervals converges to a unique point q in $q \in \mathcal{Q} := [0, 1] \times [0, 1]$ – q is defined as $h(t)$.

Theorem

$h : \mathcal{I} \to \mathcal{Q}$ defines a space-filling curve, the ***Hilbert curve****.*

Proof: h defines a Space-filling Curve

We need to prove:

- h is a mapping, i.e. each $t \in \mathcal{I}$ has a **unique** function value $h(t)$
  $\rightarrow$ OK, if $h(t)$ is independent of the choice of the sequence of intervals (see next chapter)
- $h\colon \mathcal{I} \rightarrow \mathcal{Q}$ is **surjective**:
  - for each point $q \in \mathcal{Q}$, we can construct an appropriate sequence of 2D-intervals
  - the 2D sequence corresponds in a unique way to a sequence of intervals in $\mathcal{I}$ – this sequence defines an original value of q
    $\Rightarrow$ every $q \in \mathcal{Q}$ occurs as an image point.
- h is **continuous**

Continuity of the Hilbert Curve

A function $f\colon \mathcal{I} \to \mathbb{R}^n$ is **continuous**, if

for each $\epsilon > 0$

a $\delta > 0$ exists, such that

for all $t_1, t_2 \in \mathcal{I}$ with $|t_1 - t_2| < \delta$, the following inequality holds:

$\|f(t_1) - f(t_2)\|_2 < \epsilon$

Strategy for the proof:
For any given parameters t_1, t_2, we compute an estimate for the disctance $\|h(t_1) - h(t_2)\|_2$ (functional dependence on $|t_1 - t_2|$).
$\Rightarrow$ for any given ϵ, we can then compute a suitable δ

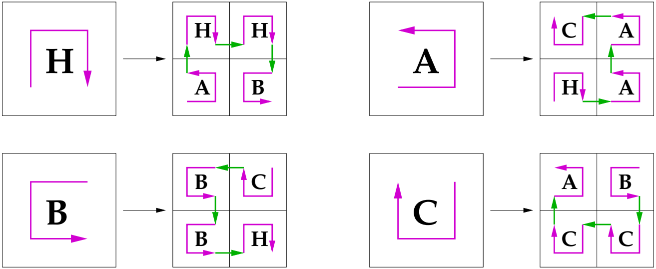
Continuity of the Hilbert Curve (2)

- given: $t_1, t_2 \in \mathcal{I}$; choose an n, such that $|t_1 - t_2| < 4^{-n}$
- in the n-th iteration of the interval sequence, all interval are of length 4^{-n}
  $\Rightarrow [t_1, t_2]$ overlaps at most two neighbouring(!) intervals.
- due to construction of the Hilbert curve, the values $h(t_1)$ and $h(t_2)$ will be in neighbouring subsquares with face length 2^{-n}.
- the two neighbouring subsquares build a rectangle with a diagonal of length $2^{-n} \cdot \sqrt{5}$;
  therefore: $\|h(t_1) - h(t_2)\|_2 \leq 2^{-n}\sqrt{5}$

For a given $\epsilon > 0$, we choose an n, such that $2^{-n}\sqrt{5} < \epsilon$.
Using that n, we choose $\delta := 4^{-n}$; then, for all t_1, t_2 with $|t_1 - t_2| < \delta$, we get: $\|h(t_1) - h(t_2)\|_2 \leq 2^{-n}\sqrt{5} < \epsilon$. Which proves the continuity!

A Grammar for Describing the Hilbert Curve

Construction of the iterations of the Hilbert curve:



$\rightarrow$ motivates a **Grammar** to generate the iterations

A Grammar for Describing the Hilbert Curve

- Non-terminal symbols: $\{H, A, B, C\}$, start symbol H
- terminal characters: $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- productions:

$$
\begin{array}{rcl}
H & \longleftarrow & A \uparrow H \rightarrow H \downarrow B \\
A & \longleftarrow & H \rightarrow A \uparrow A \leftarrow C \\
B & \longleftarrow & C \leftarrow B \downarrow B \rightarrow H \\
C & \longleftarrow & B \downarrow C \leftarrow C \uparrow A
\end{array}
$$

- replacement rule: in any word, **all non-terminals have to be replaced at the same time** $\rightarrow$ L-System (Lindenmayer)

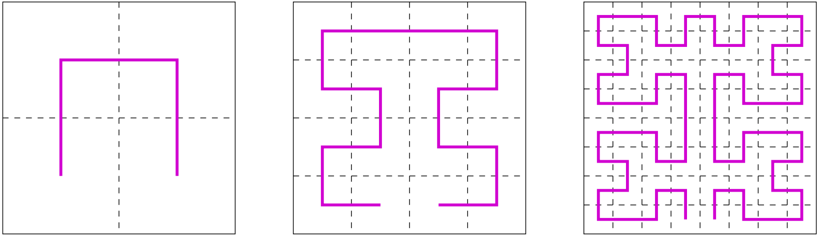
$\Rightarrow$ the arrows describe the **iterations of the Hilbert curve** in “turtle graphics”

A Grammar for Describing the Hilbert Curve

The grammar for the Hilbert curve also closes some open topics concerning its definition (and proof of continuity):

- there are only four basic patterns that occur (corresp. to the symbols $\{H, A, B, C\}$ of the grammar)
  $\rightarrow$ closed recursive system!
- two subsequent subsquares of the Hilbert-curve construction share a common edge(!)
  $\rightarrow$ follows from the fact that the move operators $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ are sufficient to describe the operators
- last but not least:
  we have formalised the construction of the iterations (replacing the inexact “rotate patters such they fit together”)

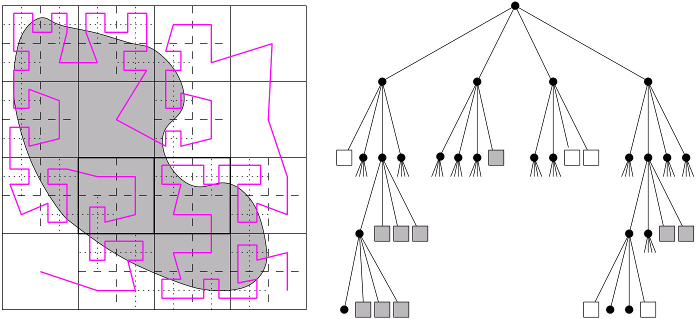
Construction of the Hilbert-Moore Curve



New Construction:

- modified orientation of the subcurves in the first iteration
- leads to a closed curve: start and end point at $\left(0, \frac{1}{2}\right)$

Hilbert Order on a Quadtree – Revisited



- traverse quadtree grid cells in Hilbert order
- relate to depth-first traversal of the quaternary tree

Remember: Grammar for the Hilbert Curve

- Non-terminal symbols: $\{H, A, B, C\}$, start symbol H
- terminal characters: $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- productions:

$$
\begin{aligned}
H &\longleftarrow A \uparrow H \rightarrow H \downarrow B \\
A &\longleftarrow H \rightarrow A \uparrow A \leftarrow C \\
B &\longleftarrow C \leftarrow B \downarrow B \rightarrow H \\
C &\longleftarrow B \downarrow C \leftarrow C \uparrow A
\end{aligned}
$$

- replacement rule: in any word, **all non-terminals have to be replaced at the same time** $\rightarrow$ L-System (Lindenmayer)
- without replacement rule:

$$
\begin{aligned}
H &\longleftarrow A \uparrow H \rightarrow H \downarrow B \\
&\longleftarrow A \uparrow A \uparrow H \rightarrow H \downarrow B \rightarrow H \downarrow B
\end{aligned}
$$

Remember: Grammar for the Hilbert Curve

- Non-terminal symbols: $\{H, A, B, C\}$, start symbol H
- terminal characters: $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- productions:

$$\begin{array}{lcl} H & \longleftarrow & A \uparrow H \rightarrow H \downarrow B \\ A & \longleftarrow & H \rightarrow A \uparrow A \leftarrow C \\ B & \longleftarrow & C \leftarrow B \downarrow B \rightarrow H \\ C & \longleftarrow & B \downarrow C \leftarrow C \uparrow A \end{array}$$

- replacement rule: in any word, **all non-terminals have to be replaced at the same time** $\rightarrow$ L-System (Lindenmayer)
- without replacement rule:

$$\begin{array}{lcl} H & \longleftarrow & A \uparrow H \rightarrow H \downarrow B \\ & \longleftarrow & A \uparrow A \uparrow H \rightarrow H \downarrow B \rightarrow H \downarrow B \end{array}$$

$\rightarrow$ information on hierarchical levels lost!

Modified Grammar for the Hilbert Curve

- Non-terminal symbols: $\{H, A, B, C\}$, start symbol H
- terminal characters: $\{\uparrow, \downarrow, \leftarrow, \rightarrow, (,)\}$
- productions:

$$
\begin{aligned}
H &\longleftarrow (\, A \uparrow H \rightarrow H \downarrow B \,) \\
A &\longleftarrow (\, H \rightarrow A \uparrow A \leftarrow C \,) \\
B &\longleftarrow (\, C \leftarrow B \downarrow B \rightarrow H \,) \\
C &\longleftarrow (\, B \downarrow C \leftarrow C \uparrow A \,)
\end{aligned}
$$

- **no additional replacement rule**
  $\rightarrow$ context-free grammar (Chomsky type-2)
- generated strings:

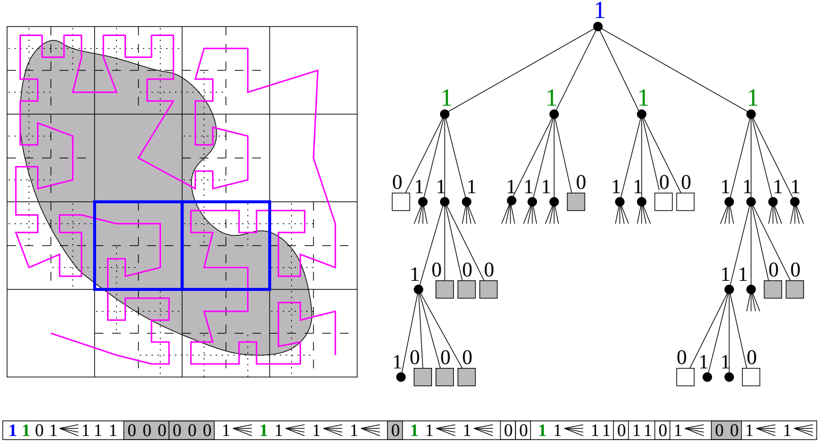
$$
\begin{aligned}
H &\longleftarrow (\, A \uparrow H \rightarrow H \downarrow B \,) \\
&\longleftarrow (\, A \uparrow (\, A \uparrow H \rightarrow H \downarrow B \,) \rightarrow H \downarrow B \,)
\end{aligned}
$$

Adaptive Hilbert Traversal – Algorithm

Algorithm 1: Scheme of an adaptive Hilbert traversal

Procedure `H()` **begin**
    if *node cell is a leaf* **then**
        `// Execute task on current position`
        `execute` (...);
    else
        `refineH` ();
        `A()`; `up()`;
        `H()`; `right()`;
        `H()`; `down()`;
        `B()`;
        `coarsenH` ();
    end
end

Bit-Stream-Encoded Quadtree and Hilbert Order



- encode quadtree as stream of “refinement bits”
- bits provided in depth-first/Hilbert order

Adaptive Hilbert Traversal – Algorithm (2)

if *node cell is a leaf* **then**
```
    // Execute task on current position
    execute (...);
```
else
```
    refineH ();
    ...;
    coarsenH ();
```
end

Check for leaf cell:

- obtain “leaf cell” encoding via bit stream
- bits in depth-first/Hilbert order: move to next bit after each check

Change of tree level:

- `refineH()`: move from parent to first child cell
- `coarsenH()`: move from last child cell back to parent